Tempe: An Interactive Data Science Environment for Exploration of Temporal and Streaming Data

Danyel Fisher, Badrish Chandramouli, Robert DeLine, Jonathan Goldstein,
Andrei Aron, Mike Barnett, John C. Platt, James F. Terwilliger, John Wernsing

Microsoft Research
Redmond, WA, USA 98052
{danyelf, badrishc, rdeline, jongold,
andreia, mbarnett, jplatt, jamest, johnwer}@microsoft.com

ABSTRACT

Over the last two decades, data scientists performed increasingly sophisticated analyses on larger data sets, yet their tools and workflows remain low-level. A typical analysis involves different tools for different stages of the work, requiring file transfers and considerable care to keep everything organized. Temporal data adds additional complexity: users typically must write queries offline before porting them to production systems. To address these problems, this paper introduces Tempe, a web application providing an integrated, collaborative environment for both real-time and offline temporal data analysis. Tempe's central concept is a persistent research notebook retaining data sources, analysis steps and results. Analysis steps are carried out in script editor that uses a live programming approach to display interactive, progressively updated visualizations.

Tempe uses a temporal streaming engine, Trill [17], as its back-end data processor. In the process of creating Tempe, we have discovered new interactivity and responsiveness requirements for Trill. Conversely, building around Trill has shaped the user experience for Tempe. We report on this cross-disciplinary design process to argue that end user experience can be an integral part of creating a data engine.

1. INTRODUCTION

Data scientists have become the primary force behind data analytics on production data, such as search logs, telemetry, and business process data. Increasingly, data scientists find that they need a broad and growing set of tools to process their data: they manage the structure of data storage; they extract it into analysis tools; they write ad-hoc scripts for data transformation and reshaping; they apply tools for statistics and machine learning; they collaborate with stakeholders and other analysts; and they explore their data and communicate their results through visualizations and descriptions of methods.

Current work practice requires a suite of tools to pull this off, with many different components. Collaborating with colleagues may mean emailing scripts, sending installation instructions, and setting data permission. Versioning often means digging through archives of old text files. It is common to explore data in a REPL ("Read-Eval-Print Loop", such as R or Python), which means that the tool does not produce a canonically correct script—and may not run again if some critical bit of state was lost.

To work in this sort of heterogeneous environment, data scientists often work with small subsets, offline; they then rewrite the code in a language that runs on a computation cluster to scale up to streaming or large scale data. The code may then be rewritten again for use in a real-time system.

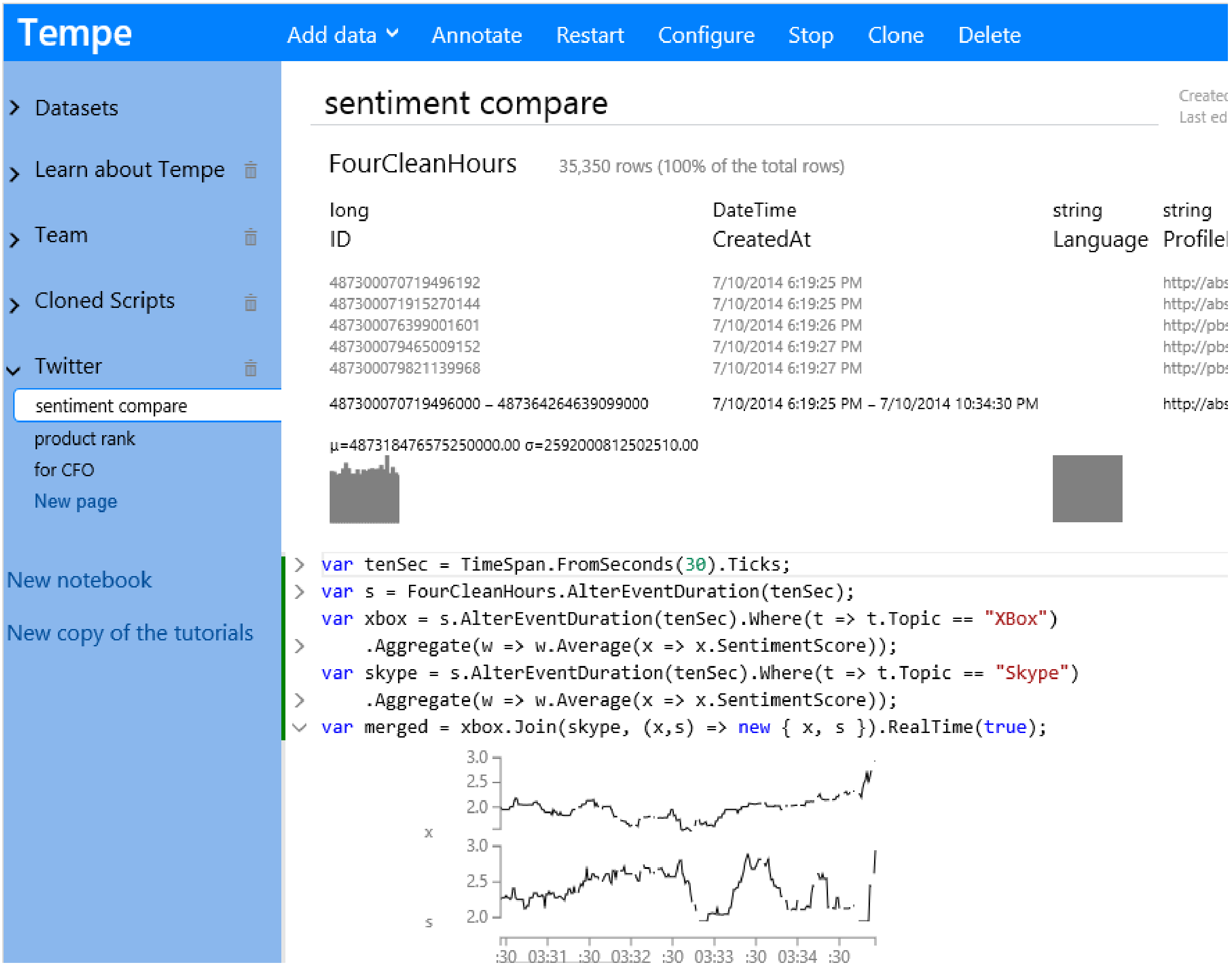
To address these many issues, this paper introduces the Tempe environment for exploratory data analysis of temporal data. Tempe is a system designed to improve both individual data scientist's work practice and their ability to share and replicate analysis results. To help individual data analytics, Tempe gathers the necessary tools for analysis within a single user experience, much as an IDE does for software development. Tempe's user experience is based on the concept of a persistent research notebook: every step a user performs and every result produced are saved automatically and kept indefinitely. This reduces the low-level clerical work of transferring, tracking, and organizing files.

With large datasets, Tempe progressively computes and reports analysis results, regardless of the size of the data sets: rather than waiting for large batch computations to complete before seeing results, a Tempe user sees instant query results that are updated once per second as additional data is processed. With streaming datasets, Tempe allows the user to interact with streamed data either offline with archived data collections, or online with live streams.

To promote sharing and collaboration between analysts, Tempe is implemented as a web application: data analysts access Tempe through a web browser, and all data handling and computation are performed on servers. As a result of this architecture, analysis artifacts are shared by default, and users all share the same computing environment and tools. Analysts can share results with colleagues by sending them a notebook page's URL; colleagues can concurrently edit a page's content to analyze data together.

Tempe was developed through a cross-disciplinary process: the creators of the front-end requested unanticipated features from the back-end data processing engine; conversely, the capabilities of the back-end inspired interesting front end features and paradigms. This "bits-to-pixels" collaboration paints an image of a highly cross-disciplinary future for systems software development.

This paper makes two contributions. First, we describe the design and implementation of Tempe, an integrated, collaborative web application for temporal and large-scale data analysis,



sentiment compare

FourCleanHours 35,350 rows (100% of the total rows)

long ID	DateTime CreatedAt	string Language	string Profilel
487300070719496192	7/10/2014 6:19:25 PM		http://abs
487300071915270144	7/10/2014 6:19:25 PM		http://abs
487300076399001601	7/10/2014 6:19:26 PM		http://pbs
487300079465009152	7/10/2014 6:19:27 PM		http://pbs
487300079821139968	7/10/2014 6:19:27 PM		http://pbs
487300070719496000 – 487364264639099000	7/10/2014 6:19:25 PM – 7/10/2014 10:34:30 PM		http://abs
μ=487318476575250000.00 σ=2592000812502510.00			

```
var tenSec = TimeSpan.FromSeconds(30).Ticks;
var s = FourCleanHours.AlterEventDuration(tenSec);
var xbox = s.AlterEventDuration(tenSec).Where(t => t.Topic == "XBox")
    .Aggregate(w => w.Average(x => x.SentimentScore));
var skype = s.AlterEventDuration(tenSec).Where(t => t.Topic == "Skype")
    .Aggregate(w => w.Average(x => x.SentimentScore));
var merged = xbox.Join(skype, (x,s) => new { x, s }).RealTime(true);
```

Figure 2. A browser screenshot of Tempe, showing a notebook page with an offline Twitter analysis.

with progressive data queries and pervasive data visualization. Second, we discuss how the process of fulfilling these design goals made for interesting cross-disciplinary collaboration.

2. SCENARIO

To illustrate Tempe's user experience, we describe a familiar scenario of a data scientist processing a Twitter feed. Tom is working with a public relations team that is tracking their Twitter traffic and wants to set up a dashboard for a handful of their critical metrics. While his team knows a handful of the metrics that they want to present, such as volume of tweets and sentiment on certain topics, they also want to investigate possible correlates of these issues by looking at past performance.

To get started with his analysis, he visits http://tempe in his web browser. On the left side of the page, he sees his collection of analysis notebooks and notebook pages (Figure 1). Under the "Twitter" notebook he clicks "New page" and titles it.

Tom has a stored archive from the Twitter gardenhose from the last twenty-four hours in CSV format. When he adds it as a data source, Tempe defines a schema for the data and generates a class for reading data from that source: for delimited files, like CSV or TSV, Tempe scans the file to define its own schema, choosing column names from a header row (if present) and column types based on the format of the data. He can, of course, manually override any choice.

To help an analyst's productivity, Tempe also automates common tasks. When a data source is added to a notebook page, Tempe asynchronously scans the data to provide useful facts about each column, such as how many data are missing or badly formatted; whether the data is sorted; how the data is distributed. Tom can now proceed with his analysis. Tempe's editor provides immediate evaluation feedback, in addition to typical editing features like code completion and parameter prompts. After Tom enters the first statement, Tempe immediately displays a table below the statement. This table is a *progressive visualization* of the evaluation result. At first, this table shows only a header row, representing the type of the result. Then, at one second intervals, the table is updated to show newly computed rows and the total number of rows computed so far. The table visualization provides navigation controls to allow Tom to browse through the results.Tom can now proceed with his analysis. Tempe's editor provides immediate evaluation feedback, in addition to typical editing features like code completion and parameter prompts. After Tom enters the first statement, Tempe immediately displays a table below the statement. This table is a *progressive visualization* of the evaluation result. At first, this table shows only a header row, representing the type of the result. Then, at one second intervals, the table is updated to show newly computed rows and the total number of rows computed so far. The table visualization provides navigation controls to allow Tom to browse through the results.

While the data is flowing fine for his search for "Skype", a second search for "XBoz" isn't returning any results. Tom realizes that he had a typo in his query and corrects it. The display updates instantly to show search results. Tempe's "live programming" experience allows any statement to be edited at any time. Tempe cancels any obsolete running queries and selectively recomputes queries to bring the onscreen results up to date.

Jane, one of Tom's colleagues, wants to connect his analysis to the live Twitter feeds coming to their system. Jane has a small C# library that connects directly to the Twitter stream. Jane goes to the URL for Tom's page, and presses *Clone* to generate a new copy of the page with the same analysis. She presses *Configure* and add her library to the script. She adds a few lines of code to instantiate her Twitter observer class and turn it into a stream of the same type as the offline data. With this new live stream, she is able to reuse the remaining code in Tom's script. The script automatically updates to start reading this live streaming data. Now, anyone who visits her page sees the latest results of the telemetry analysis.

3. BACKGROUND AND RELATED WORK

3.1 Studies of data analysts

There have been relatively few studies of the work practices of data analysts. Kandel *et al.* [9] interviewed business intelligence workers and identified three different personas: "hackers", who are comfortable writing code and using a wide variety of tools in a variety of different languages; "scripters", who prefer to use structured data but create more sophisticated data models; and "application users", who tend to use prepared data within dedicated analysis applications. Fisher *et al.* [7] interviewed data analysts about their workflows and documented their pain points, for example, transferring data is slow and cumbersome, scripts run slowly and non-interactively, and it can be difficult to reshape data between the different file formats required by different tools. They describe an analyst's typical workflow as winnowing large amounts of low-value data down to small amounts of high-value data: from filtering and aggregating, for example, using a map/reduce system; to modeling, using machine learning and statistical tools; to plotting, charting or reporting the final results of the analysis.

3.2 Tools for data analysis

The recent excitement in the press about "big data" or "big data analytics" has been made possible by the current generation of map-reduce tools [5], like Hadoop[1], running on large clusters of PCs. Many data analysts today implement map-reduce computations by using scripting notations like Sawzall [14] or Pig Latin [13]. These notations allow an analyst to express analyses in terms of high-level relational queries. Tempe uses a data processing library, called Trill, which supports temporal data queries, programmed through a high-level scripting language. Tempe scripts use types derived from the schemas of the imported data sources. Finally, the goal of Tempe is also to cover complete analysis workflows, including modeling, statistical tests, and visualization.

Many data analysis tools use a research notebook metaphor, going back to the early versions of Mathematica[2]. Today, many data analysts are adopting the IPython Notebook[3], which provides an interactive Python interpreter with embedded visualizations. Although the IPython Notebook uses a web browser as its front end, it is a desktop application rather than a web application. Users, however, can share non-interactive versions of their notebook pages on the web. Tempe provides a similar scripting experience to these notebooks, but as a web application. In contrast to the IPython Notebook, Tempe users share fully interactive notebook pages, with no need to install software locally. Wakari[4] is a new product that hosts IPython Notebook's in Amazon EC2, which is closer to a central web application, but silos each user's interactive notebook behind a separate pay wall.

Tempe also provides a different live programming experience than the IPython Notebook or Mathematica. In these other tools, the notebook's content is treated as view on an interpreter session, which makes it possible to create notebook pages that look incorrect. For example, if the user edits a notebook page like this:

```
x = 3;          x is 3
~~x = x + 1;~~
y = x;          y is 4
```

(that is, an increment statement is evaluated, then erased), then y is bound to 4, which looks incorrect given the two remaining statements on the page. Instead, Tempe's live programming experience maintains the invariant that the evaluation results reflect the script's current content, which we feel is less confusing.

One aspect of a collaborative research is tracking workflow provenance: the ability to know how data flows into results. There have been both community efforts and technical efforts to allow data analysts to share and replicate each other's work. The Open Provenance Model [26] is a well-known standard for coarse-grained workflow provenance. The Lipstick [25] system enables workflow provenance (in additional to traditional tuple-level database-style provenance) in Pig Latin.

3.3 Analytics Systems

A number of big data analytics systems that support interactivity and exploration have been proposed in the database community over the last decade. The CONTROL project [23] from Berkeley resulted in seminal systems such as Potter's Wheel [22] that provided better UX integration with support for approximate quick answers, but the focus was on building constrained interfaces (such as drop-down boxes and spreadsheet-style interfaces) that support online aggregation. More recently, there have been several efforts to make databases more usable with better integration with the rest of the UI: For example, dbTouch [20] supports a touch friendly interface to databases, not focused on developing logic, targets traditional DB back-end. QWiK [11] and DICE [21] are visualization front-ends

[1] http://hadoop.apache.org
[2] http://www.wolfram.com
[3] http://ipython.org/notebook.html
[4] http://www.wakari.io

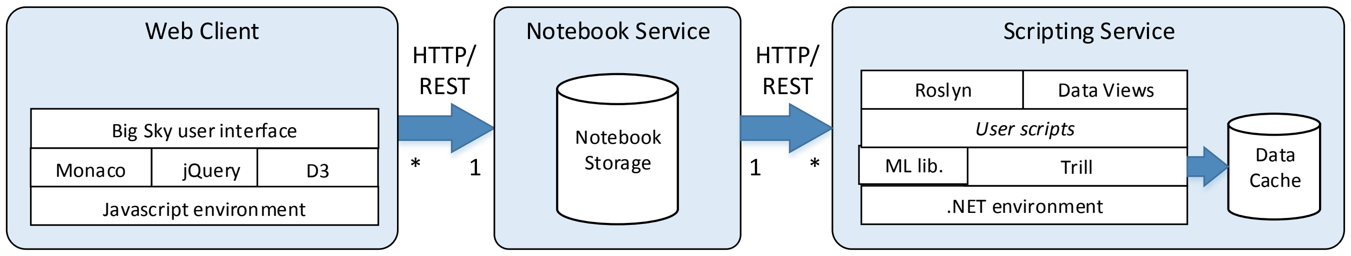


Figure 3. Tempe's architecture has three components: a Web Client that provides the user experience in a web browser; a Notebook Service that stores the user-generated content; and a Scripting Service that provides a computing environment for user scripts.

targeted towards OLAP style data cube exploration, UX integration enables pre-computation and fetches results in advance, supports approximate progressive results.

In the big data ecosystem, SciDB [18] focuses on disk resident array based data, and provides high-performance with scaled out execution. They do not target real time queries, and their temporal support is limited to traditional time-series. ScalaR [19] is a visualization front-end that can work against SciDB as well as other databases (they argue for decoupling the front-end from the back-end). ScalaR provides visual querying capabilities, but is not a REPL: they do not focus of the problem of developing logic. Their current focus is on geographic and map data. BlinkDB [24] supports interactive SQL queries (not real-time or temporal) on large datasets, but does not focus on visualizations.

4. IMPLEMENTATION

4.1 System Architecture

Tempe is structured in three major components (Figure 2): the Web Client implements the user experience; the Notebook Service stores the user-generated content and multiplexes requests from multiple clients; and the Scripting Service provides the computing environment for the user's scripts.

The Web Client is implemented in Javascript and uses several existing components: jQuery[5], a popular user interface toolkit for the web; Monaco[6], a source code editor, with functionality similar to that of Microsoft Visual Studio or Eclipse, including syntax highlighting and code completion; and D3.js [3]. The Web Client is "thin": it performs no data analysis computations, but instead handles user interface events and invokes the Notebook Service's REST-style interface [6].

The Notebook Service is a standard stateless web server responsible for storing and retrieving user-generated content, like notebooks, notebook pages, data sources, and scripts. The Notebook Service delegates changes to the user's scripts to the Scripting Service. It also stores the results that service returns, so that analysis results, like tables and charts will be preserved indefinitely.

The Scripting Service provides the computing environment for scripts. It retains the state of a script's ongoing computation. The Scripting Service is implemented using Microsoft Roslyn [12], which provides an interpreter for the C# language. The language that Roslyn interprets is a version of C# in which source code can arbitrarily mix top-level definitions (classes, structs, enums), method definitions, and statements. This mixture makes the C# scripting language similar to other scripting languages for data analysis, like R, MATLAB, and Python. The Tempe UI allows the user to link her choice of libraries to the Roslyn interpreter session.

4.2 Live Programming

The heart of the Tempe user experience is the live programming of scripts. Most data analysts are familiar with a read-eval-print loop (REPL) experience, like that in R or Python, in which the user iteratively enters a statement and waits to see a printed response. To see alternative results, the user often re-enters previous statements, making the necessary changes. In contrast, Tempe's live programming experience allows the user to edit any part of the script text at any time, as she would in a development environment. Tempe automatically re-evaluates statements as necessary to bring the result visualizations up to date with the text.

The implementation of live programming involves all three components. The Web Client's editor captures the user's edits, which are then sent to the Notebook Service for storage. Since a script can be simultaneously edited by multiple people using multiple Web Clients, the Notebook Service combines edits from all Web Clients into a coherent script content. The Notebook Service reports the new script content to the Scripting Service.

The Scripting Service first parses the new script content into a syntax tree, reporting any errors. It then uses the well-known Levenshtein edit distance algorithm to find a minimal sequence of edits to transform the old sequence of top-level statements into the new sequence. Based on these edits, it first cancels any obsolete ongoing queries. It then submits to the Roslyn interpreter both the changed statements and any subsequent statements with data- or control-flow dependencies on these changed statements. The computation of dependencies is a conservative approximation, since a precise dependency graph is undecidable.

Tempe uses the model-view-controller pattern [8] to visualize execution results. Each statement that the Roslyn interpreter executes results in a (potentially huge) .NET object. The Scripting Service creates a model of each .NET object, which it reports back to the Notebook Service for storage. The Notebook Service in turn reports the model to the Web Client, whose interactive visualizations act as the view and controller. The nature of the model depends on the user's choice of view. For example, for a table view, the model is a sample of rows; for a histogram view

[5] http://www.jquery.com

[6] Monaco has not yet been publicly released.

of categorical data, the model is an aggregated count per categorical value. In all cases, the model is of a bounded size, regardless of the size of .NET object it summarizes.

4.3 Data Processing

For the parts of a script that perform queries over data, Tempe uses Trill [17], a temporal streaming engine named for its goal of querying a trillion (10^{12}) events per day. A description of Trill is outside the scope of this paper (see [17] for details), but we note that Trill is capable of high-performance streaming analytics across the latency spectrum from offline (such as historical logs) to real-time. Tempe uses Trill to handle temporal queries – both on real-time streams and offline logs. In addition, Tempe takes advantage of Trill's temporal processing capability to answer *non-temporal* queries progressively (with early results), by reusing Trill's notion of time to instead mean query computation progress [4]. A progressive query provides partial answers while it is computing, as opposed to batch systems that do not provide any feedback until the query has completed.

To keep track of progressive/temporal updates, the Notebook Service polls the Scripting Service once per second for the latest models, which are then pushed to any relevant Web Clients. This design decouples three time rates — the rate at which query results are computed, the rate at which results are summarized, and the rate at which results are reported to the user. This allows us to control the balance between latency (reporting updates quickly) and throughput (computing results in large batches). The Notebook Service polls for progressive updates until they are complete; it polls for temporal queries until the user explicitly stops them.

5. CROSS-DISCIPLINARY, BITS-TO-PIXELS

The architecture discussed here comes about from the unusual structure of the Tempe project: we chose to scope the project "from bits to pixels". The tasks in a data analyst's workflow encompass a broad set of technical problems, including accessing, organizing, transforming and visualizing data. To address these problems, we brought together a diverse team. Our collaboration includes experts in user interfaces, visualization, programming languages, machine learning, and data systems. We also worked closely with users, bringing out early prototypes frequently to data scientists and incorporating data scientists into the team as participant experts. To take the best advantage of this broad expertise, we created our own software for any component where we had innovative ideas—like temporal query processing, live programming and progressive visualization—and otherwise reused large components—like the C# interpreter and program editor.

In this section, we outline some of the ways that the design of both the Tempe system and the underlying Trill query engine were affected by each other's design, constraints, and interaction with users.

5.1 Progressive and Temporal Computing

The idea of progressive computation has been proposed in the research community for a number of years. The CONTROL [23] project, for example, showed that progressive results could lead to a more responsive data environment, even as data scales grew arbitrarily. We decided to experiment using a streaming engine to implement a progressive computation.

When we brought Trill in as part of the Tempe project, Trill was based on a notion of temporal data seen as segments with paired start- and end-events. In a progressive computation, however, the stream consists only of start-events: elements are never removed from the stream. We were able to optimize Trill for this scenario by creating special operators that work efficiently over streams with no end-events. This property also allowed Tempe to easily implement progressive data tables.

The user interface can also take advantage of the Trill engine: "progress" is a first-class notion of Trill, and Tempe directly exposes progress reports to the user. Tempe can show precisely how much of the input goes into the current progressive state.

The nature of progressive computation, in turn, implies that Tempe have an asynchronous scripting experience. Unlike standard REPLs—in which the process fully evaluates a query, then prints the results—Tempe provides continuous updates, even as the user continues to the next query. Data results become available when they are ready.

Performance Considerations: Designing Trill as a Tempe component meant that we needed to ensure rapid computation for large data jobs. Streaming engines can be slow, though, as they process individual items and produce individual output. Knowing that Trill's front-end would be in the Tempe system, we realized that we could accept latency on a human time scale: the user interface only looks for updates once a second. Trill batches data elements, trading off throughput for latency in order to maintain responsiveness.

In exploratory work, data scientists often change their minds, which means that many queries quickly become obsolete. This context allows us to accelerate query cleanup. While earlier versions of Trill would carefully clean up memory resources, Tempe simply kills the Trill thread, allowing the OS to take care of memory cleanup, at a substantial savings.

Temporal Computation: Trill's implementation of progressive computation actually creates a continuous stream of events. With this engine, it was straightforward to incorporate temporal computations directly into the interface with few changes. Because Tempe sees a disk-based stream the same way it sees a live stream, providing a consistent user experience for both online and offline streams also came easily.

5.2 Stream Properties and Data Inference

Interface requirements and the core of the query engine interacted closely in tracking stream properties. There are some types of data that can be useful for the engine, but are not usually solicited from the user. For example, if the fact that a column is known to be categorical—to have a small cardinality—than the engine can make decisions about compression, grouped aggregation, and scale out. This data is often hard to acquire, requiring an extra pass across the column.

Data scientists, however, also wants to know about cardinality: it informs the analyses they choose to do and affects the sorts of visualizations they draw on the data. As a result, the user has a strong interest in helping label input columns as categorical—or correcting the built-in inference system if it chooses incorrectly. Categorical labels are then carried along as stream properties, propagated through queries, and are fed both into the interface and the engine, enhancing both experiences.

Similar to the categorical case, we are currently implementing a facility to label a stream as a "signal." A signal means that no more than one value is defined for a given key at each time. Again, this label benefits both the user and the engine. If we know that a stream is a signal, we can visualize it in a more intuitive form (e.g. line charts) and allow direct arithmetic computations in the user interface (e.g., divide one stream by another). We can also feed cues into the engine to optimize the method of storage for signals.

Optimizing on Start-Edge Only: Trill ordinarily thinks of data as segments with paired start- and end-events; although it is capable of handling start-edge-only. With Tempe, we found many use cases of progressive queries, where the stream consisted only of start-events, i.e., elements are never removed from the stream. We were able to optimize Trill for this scenario by creating special operators that work efficiently over streams with no end-events. This property also allowed Tempe to easily implement Now optimized for start-edge-only, which was necessary to implement progressive data tables.

Batched Computation: Knowing that the UI will require data only so often—currently once a second—means that the computation system can afford to batch up computations, as long as it is at least as responsive as the UI. User can choose batch size—smaller batches for better latency (at the possible cost of some throughput); larger batches vice versa.

5.3 Discussion

In this section, we have outlined several technical advantages gained by close collaboration between the front-end and back-end. Both sides have gained from this knowledge; designing from “bits to pixel” has taught us a tremendous amount about the opportunities available. More importantly, when it comes to implementation decisions, having direct access to users changes our priorities and design.

That said, this research approach requires care to keep from “boiling the ocean”. In several cases, we were able to implement our ideas as increments to existing mature components. We implemented live programming while reusing an existing compiler/interpreter infrastructure; we implemented scripts as rich media while reusing an existing code editor; and we implemented progressive visualization using an existing visualization toolkit. On the other hand, we ignored some popular existing technologies because they would have preemptively blocked our innovations. In particular, we rejected the Hadoop-based big-data stack in favor our own Trill temporal data engine. Similarly we avoided the R interpreter, with its unsafe C-based execution model in favor of C# and .NET. This means that we have missed out on many of the community tools and easier adoption path.

6. CONCLUSIONS

This paper introduces the Tempe integrated environment for temporal data analyses. Tempe addresses two problems. For individual analysts, it supports complete workflows within a simple user interface, obviating the need to switch between different tools and to manage data interchange between them. For research communities, Tempe’s web application architecture makes sharing and retention of analyses the default.

The creation of Tempe came from a productive collaboration between user interface and database researchers. We hope to see more such cross-disciplinary collaborations in the future.

7. REFERENCES

[1] Barnett, M., Chandramouli, B., DeLine, R., Drucker, S., Fisher, D., Goldstein, J., Morrison, P., and Platt, J. “Stat! – An Interactive Analytics Environment for Big Data”, *Proc. of the 2013 ACM SIGMOD International Conference on Management of Data (demonstration)*, pages 1013-1016.

[2] Bevan, J., Whitehead, E. J., Jr., Kim, S., and Godfrey, M., “Facilitating software evolution research with Kenyon,” In *Proc. ESEC/FSE*, 2005.

[3] Bostock, M., Ogievetsky, V., and Heer, J., “D3: Data-Driven Documents”, *IEEE Trans. Visualization & Comp. Graphics* (*Proc. InfoVis*), 2011.

[4] Chandramouli, B., Goldstein, J., and Quamar, A. “Scalable Progressive Analytics on Big Data in the Cloud.” *Proc. Intl. Conf. on Very Large Data Bases* (*VLDB '14*), 2014.

[5] Dean, J. and Ghemawat, G., “MapReduce: simplified data processing on large clusters,” in OSDI, 2004

[6] Fielding, R. T. and Taylor, R. N., “Principled Design of the Modern Web Architecture”, *ACM Transactions on Internet Technology* 2(2): 115–150.

[7] Fisher, D., DeLine, R., Czerwinski, M., and Drucker, S., “Interactions with Big Data Analytics”, *ACM Interactions*,, May 2012

[8] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design *Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[9] Kandel, S., Paepcke, A., Hellerstein, J.M., and Heer, J. “Enterprise Data Analysis and Visualization: An Interview Study,” In. *Proc. IEEE Visual Analytics Science & Technology* (*VAST*), Oct 2012.

[10] Meijer, E., “The World according to LINQ”, *ACM Queue* 9:8, Aug. 2011.

[11] Arnab Nandi: Querying Without Keyboards: CIDR 2013

[12] Ng, K., “The Roslyn Project: Exposing the C# and VB compiler’s code analysis”, http://www.microsoft.com/en-us/download/details.aspx?id=27744

[13] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A., “Pig Latin: a not-so-foreign language for data processing,” in *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 2008

[14] Pike, R., Dorward, S., Griesemer, R., and Quinlan, S., “Interpreting the data: Parallel analysis with Sawzall,” Sci. Program., vol. 13, no. 4, 2005.

[15] Pérez, F., and Granger, B.E., “IPython: A System for Interactive Scientific Computing”, *Computing in Science and Engineering* 9:3, May/June 2007

[16] Rowstron, A., Narayanan, D., Donnelly, A., O’Shea, G., Douglas, A., “Nobody ever got fired for using Hadoop on a cluster”. In *1st International Workshop on Hot Topics in Cloud Data Processing,* 2012.

[17] B. Chandramouli et al. The Trill Incremental Analytics Engine. MSR Technical Report. http://aka.ms/trill-tr.

[18] P. Cudre-Mauroux et al. A Demonstration of SciDB: A Science-Oriented DBMS. In VLDB, 2009.

[19] L. Battle et al. Dynamic Reduction of Query Result Sets for Interactive Visualization. In IEEE BigDataVis Workshop, 2013.

[20] Stratos Idreos and Erietta Liarou. dbTouch: Analytics at your Fingertips. In CIDR, 2013.

[21] N. Kamat et al. Distributed and Interactive Cube Exploration. In ICDE 2014.

[22] V. Raman and J. Hellerstein. Potter’s Wheel: An Interactive Data Cleaning System. In VLDB, 2001.

[23] R. Avnur et al. CONTROL: Continuous Output and Navigation Technology with Refinement On-Line. In SIGMOD, 1998.

[24] S. Agarwal et al. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In ACM EuroSys 2013.

[25] Y. Amsterdamer et al. Putting Lipstick on Pig: Enabling Database-style Workflow Provenance. In VLDB, 2012.

[26] L. Moreau et al. The Open Provenance Model: An overview. In IPAW, 2008.

8. DEMO WALKTHROUGH

We will bring an instance of the Tempe system as a demonstration for the conference. In our demonstration, we will be able to show major interactive features of Tempe, including live coding, sharing sheets and datasets, and both online and offline temporal datasets. We will demonstrate the use of Tempe against both a live stream of data from Twitter, and from a smaller offline dataset. In each of these demonstrations, we will:

• Load a notebook page to see archived results of the last time the query was run

• Interactively modify the demonstration queries, such as altering keywords or filters, and see updated results based on the modified query

• Share results with another user running a second browser

Each of these demos will be kept in the same research notebook; therefore, they can be executed at will.

8.1 Demo 1: Twitter Data Exploration and Fusion

The first demo shows how Tempe can be used to explore temporal data online and offline. We will begin with on a large sample of Twitter data. During the course of the demo, we will begin with one file containing sentiment keywords, and then start a continuous stream from a second large file containing tweets. We will join the stream with the sentiment scores, and show dynamically create a histogram of distributions of sentiment on the tweet stream. We will produce several visualizations based on this dataset. The dataset is temporal, and so we can show how the data changes over time, and will draw line charts showing the data varying.

We will then adapt the script we generated for this offline data to an online data stream; if the network cooperates, we will change from watching the query on archived data to seeing it on current data.

Figure 1, and the scenario above, are aspects of this demonstration.

8.2 Demo 2: Top-k Search Correlation

The second demo is a non-temporal data query called "top-k search correlation" that operates over a sample of a 10TB search log dataset. This query was created by users working on a feature discovery task for a search engine. The query is is run as a series of interactive LINQ expressions in the Trill environment.

The user provides a keyword (e.g., "music") as input parameter, and the sequence of queries returns the top-k words most closely correlated with the provided term. This is a two stage process. The first stage uses the search data set as input and partitions by user. For each user, we compute a histogram that reports, for each word, the number of searches with and without the input term, and the total number of searches. The second stage job groups by word, and aggregates the histograms from the first stage, computes a per-word goodness metric, and applies a top-k operation to report the k highly correlated words to the input term.

The query is expressed in LINQ and executed incrementally by Trill in the backend. We show that the results converge on the first few results quickly, allowing the user to make decisions even after looking at a tiny percentage of the data.

8.3 Visitor Interactivity

Both of these demos are fully interactive. Visitors to the demo can broadly update or modify the demonstration code. At simplest, they can change parameters to the top-k search correlation query and visualize these results interactively. They can also tweak the query—altering its functions or its calls—to learn about both the iterative exploration process and the debugging experience.

Similarly, users will be able to alter the Twitter experiment, including choosing new visualizations, adding or removing keywords, and otherwise tweaking the query.

We will bring several other data sources so that users can issue and visualize their own interactive queries.